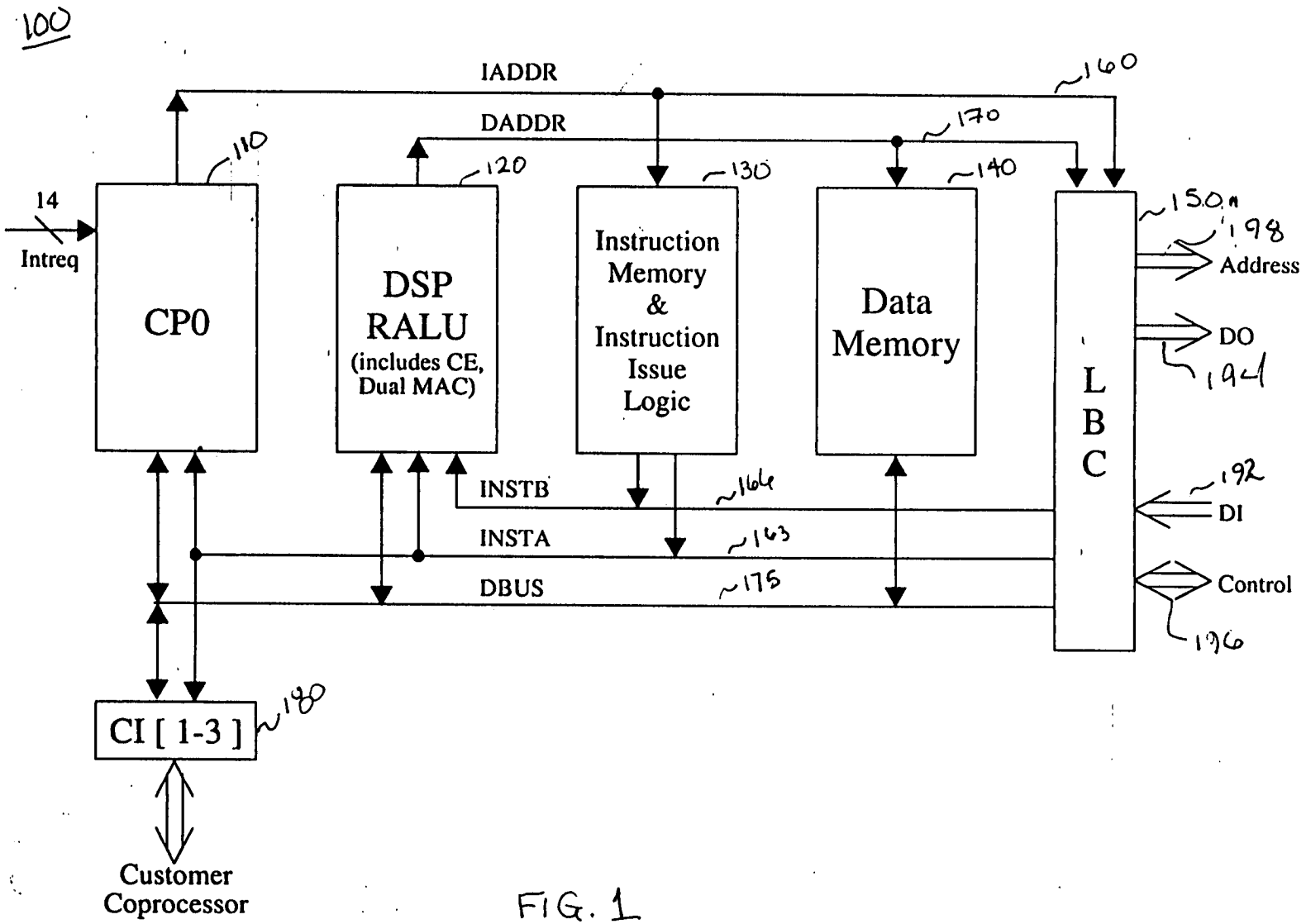

100
IADDR
160
DADDR
170
110
120
130
140
150
14
Intreq
CP0
DSP
RALU
(includes CE,
Dual MAC)
Instruction
Memory
&
Instruction
Issue
Logic
Data
Memory
L
B
C
198
Address
DO
194
INSTB
164
INSTA
163
DBUS
175
192
DI
Control
196
CI [1-3]
180
Customer
Coprocessor

FIG. 1

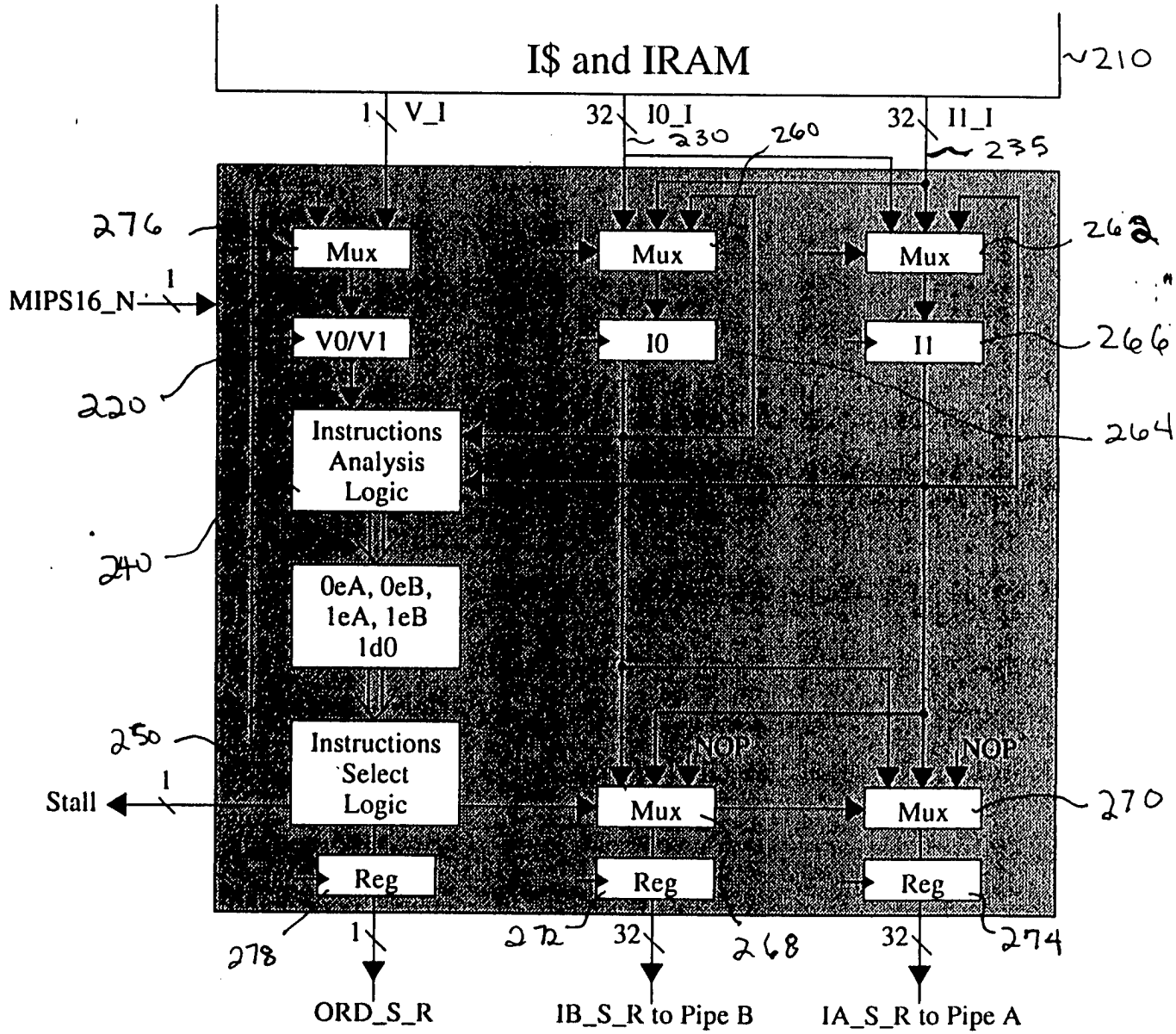

I$ and IRAM
210
1
V_I
32
I0_I
230
260
32
I1_I
235
276
Mux
Mux
Mux
262
MIPS16_N
1
V0/V1
I0
I1
266
220
Instructions Analysis Logic
264
240
0eA, 0eB, 1eA, 1eB 1d0
250
Instructions Select Logic
NOP
NOP
Stall
1
Mux
Mux
270
Reg
Reg
Reg
278
1
272
32
268
32
274
ORD_S_R
IB_S_R to Pipe B
IA_S_R to Pipe A

FIG 2

: Instruction Select Logic

300

		0eA : 0eB			
		00 (I0 not valid)	**01**	**11**	**10**
1eA : 1eB	00 (I1 not valid)	IA←NOP IB←NOP ORD←d/c V[1:0]←next	Not possible	Not possible	Not possible
	01	IA←NOP IB←I1 ORD←B V[1:0]←next	IA←NOP IB←I0 Stall←1 ORD←B V[1:0]←01	IA←I0 IB←I1 ORD←A V[1:0]←next	IA←I0 IB←I1 ORD←A V[1:0]←next
	11	IA←NOP IB←I1 ORD←B V[1:0]←next	IA←I1 IB←I0 ORD←B V[1:0]←next	IA←I0 IB←I1 ORD←A V[1:0]←next	IA←I0 IB←I1 ORD←A V[1:0]←next
	10	IA←I1 IB← NOP ORD←A V[1:0]←next	IA←I1 IB←I0 ORD←B V[1:0]←next	IA←I1 IB←I0 ORD←B V[1:0]←next	IA←I0 IB←NOP Stall←1 ORD←A V[1:0]←01
1d0		Not possible	IA←NOP IB←I0 Stall←1 ORD←B V[1:0]←01	IA←I0 IB←NOP Stall←1 ORD←A V[1:0]←01	IA←I0 IB←NOP Stall←1 ORD←A V[1:0]←01

* Note: I0 valid and I1 not valid won't occur because there is no out of order execution.

FIG.3

09637500 .082100

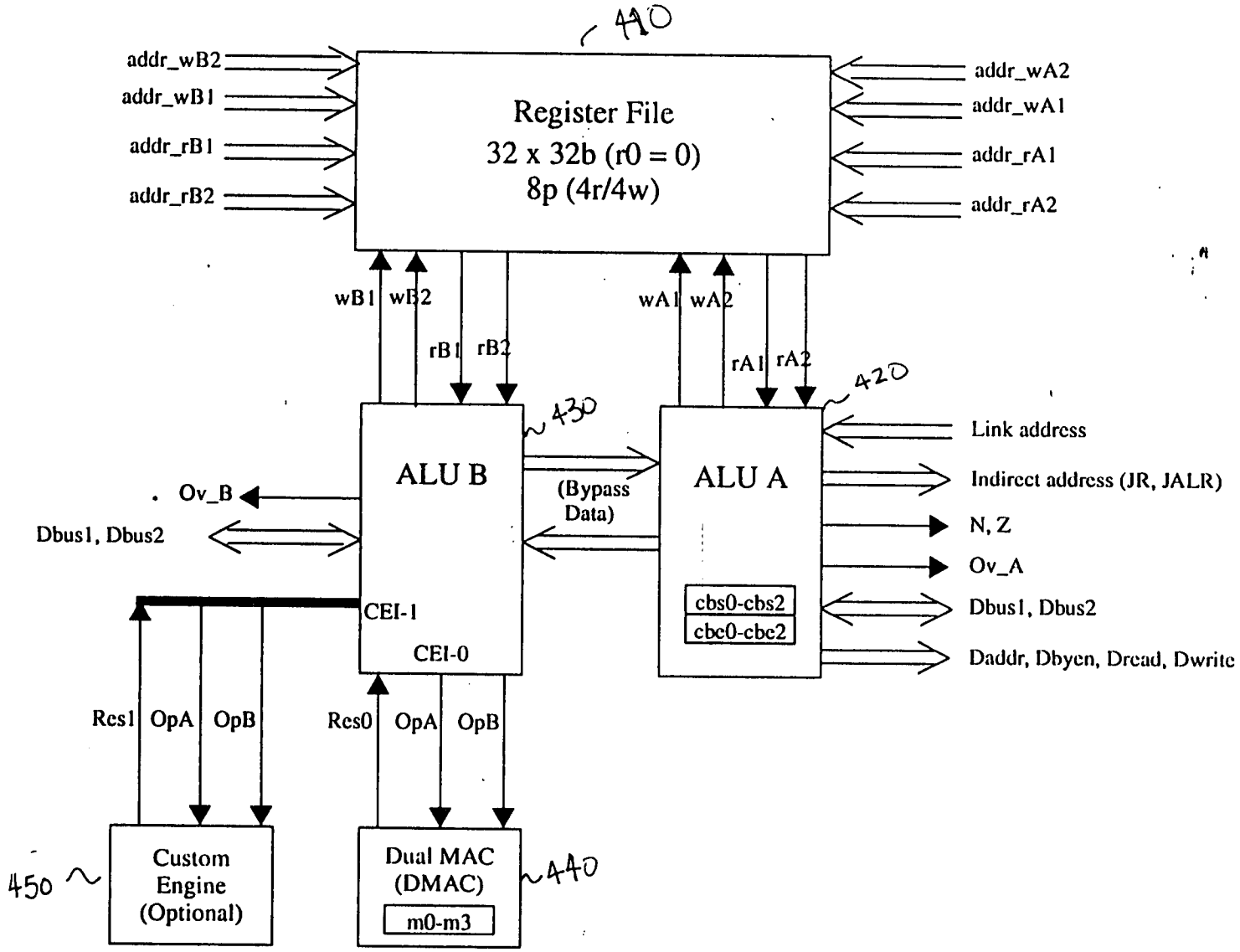

400
410
addr_wB2
addr_wB1
addr_rB1
addr_rB2
Register File
32 x 32b (r0 = 0)
8p (4r/4w)
addr_wA2
addr_wA1
addr_rA1
addr_rA2
wB1
wB2
rB1
rB2
wA1
wA2
rA1
rA2
420
430
ALU B
ALU A
Link address
Indirect address (JR, JALR)
N, Z
Ov_A
Dbus1, Dbus2
Daddr, Dbyen, Dread, Dwrite
Ov_B
Dbus1, Dbus2
(Bypass
Data)
CEI-1
CEI-0
cbs0-cbs2
cbc0-cbc2
Res1
OpA
OpB
Res0
OpA
OpB
450
Custom
Engine
(Optional)
Dual MAC
(DMAC)
m0-m3
440

FIG. 4

MMD (Radiax User Register 24)

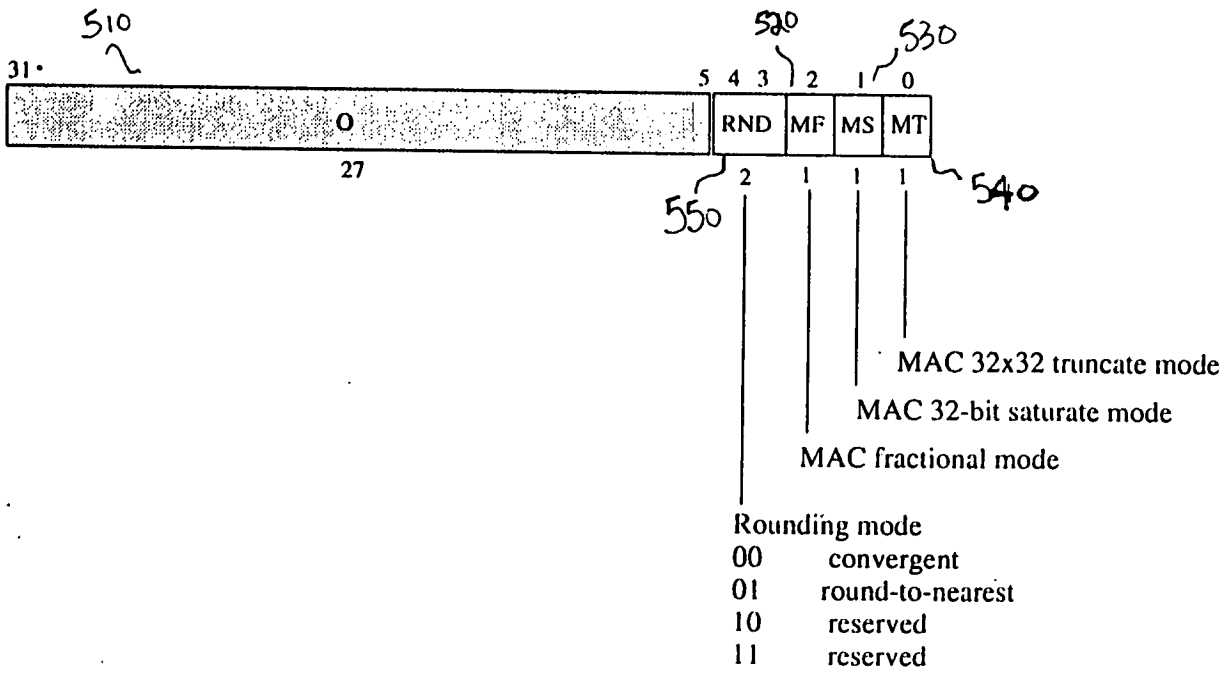


FIG. 5

09637500 .082100

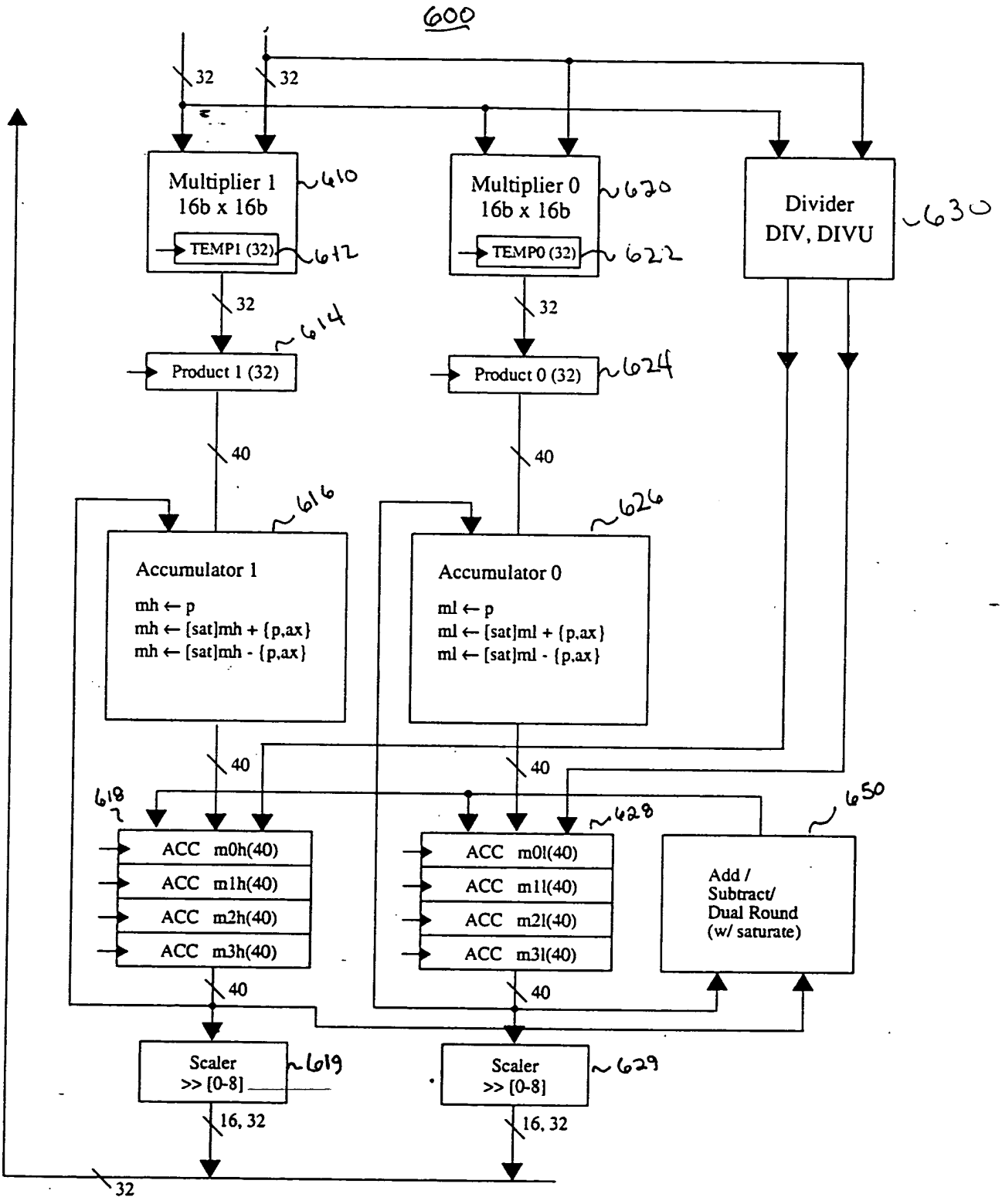

600
32
32
Multiplier 1
16b x 16b
610
TEMP1 (32)
612
Multiplier 0
16b x 16b
620
TEMP0 (32)
622
Divider
DIV, DIVU
630
32
32
614
Product 1 (32)
Product 0 (32)
624
40
40
616
626
Accumulator 1
mh ← p
mh ← [sat]mh + {p,ax}
mh ← [sat]mh - {p,ax}
Accumulator 0
ml ← p
ml ← [sat]ml + {p,ax}
ml ← [sat]ml - {p,ax}
40
40
618
628
650
ACC m0h(40)
ACC m1h(40)
ACC m2h(40)
ACC m3h(40)
ACC m0l(40)
ACC m1l(40)
ACC m2l(40)
ACC m3l(40)
Add /
Subtract/
Dual Round
(w/ saturate)
40
40
Scaler
>> [0-8]
619
Scaler
>> [0-8]
629
16, 32
16, 32
32

FIG. 6

- Integer format interpretation:

X =

S	14	13	12			---	0

^
(radix point)

$$X = -2^{15} \times S + \sum_{0}^{14} X[i]2^{i}$$

Product = P[31:0] is sign-extended to P[39:0] = $P[31]^8$ || P[31:0] for accumulation

P =

S	30	29	28			---	0

^
(radix point)

$$P = -2^{31} \times S + \sum_{0}^{30} X[i]2^{i}$$

- Fractional format interpretation:

X =

S	14	13	12			---	0

^
(radix point)

$$X = -1 \times S + \sum_{0}^{14} X[i]2^{i-15}$$

Product = P[31:0] is left-shifted one bit and sign-extended to:
P[39:0] = $P[30]^8$ || P[30:0] || 0 for accumulation

P =

S	29	28	27			0	zero

^
(radix point and product left shifted by one)

$$P = -1 \times S + \sum_{0}^{29} P[i]2^{i-30}$$

FIG. 7A

0963750 108100

Overflow Protection: Guard Bits and Saturation

- The LX5280 accumulator implements eight (8) guard bits to protect against overflow. The alternatives of (i) product scaling or (ii) input scaling by right shifting, cause loss of precision.

- Optional saturation (MADDA2.S, MSUBA2.S, ADDMA.S, SUBMA.S) can be used to avoid wrap-around on underflow or overflow of the 40-bit format (or 32-bits if that mode is selected in the MMD register):

  if (result > 0 1 1 1 1 ... 1) result = 0 1 1 1 1 ... 1

  if (result < 1 0 0 0 0 ... 0) result = 1 0 0 0 0 ... 0

- In 32-bit saturate mode, the MAC implements a full 40-bit saturation detector. This allows for the case where the accumulator holds a value greater than the maximum 32-bit saturated value prior to the addition (or subtraction) with saturation.

FIG 7B

MAC Output Control: Rounding and Scaling

- For output storage, the 40-bit accumulators must be converted to 16-bit or 32-bit format.
- Scaling

  single accumulator:

  $$RES[31:0] \leftarrow \{ mTh, mTl \} [31 + n : n] \qquad [n = 0 - 8]$$

  dual accumulators (select high half of each, useful for fractional arithmetic results):

  $$RES[31:0] \leftarrow mTh [31 + n : 16 + n] \parallel mTl [31 + n : 16 + n] [n = 0 - 8]$$

- To avoid the bias introduced by truncation, the accumulator can be rounded prior to output scaling (useful for fractional arithmetic results).

  $$\{ mT, mTh, mTl \} \leftarrow RNDA2 \{ mT, mTh, mTl \} \qquad [n = 0 - 8]$$

  $\Rightarrow$ the rounding mode is selectable in the MMD register

  bit position 16 + n of each accumulator in the pair is the least significant bit of precision after rounding

FIG. 7C

5963750 10031100

MAC Radiax Instruction Summary

800

Instruction	Syntax and Description
Dual Move to Accumulator	*MTA2[.G] rS, {mD, mDh, mDl}* If MTA2, and mDh(mDl) is selected, sign-extend the contents of general register rS to 40-bits and move to accumulator register mDh(mDl). If MTA2, and mD is selected, update both mDh and mDl with the 40-bit, sign-extended contents of the same rS. If MTA2.G is selected, the accumulator register bits [39:32] are updated with rS[31:24]; bits [31:00] of the accumulator are unchanged. (The .G option is used to restore the upper-bits of the accumulator from the general register file; typically, following an Exception.)
Move From Accumulator	*MFA rD, {mTh, mTl} [,n]* Move the contents of accumulator register mTh or accumulator register mTl to register rD with optional right shift. Bits [31+n : n] from the accumulator register are transferred to rD[31:00]. The range n = 0 - 8 is permitted for the output alignment shift amount. In the case of n = 0, the field may be omitted.
Dual Move From Accumulator	*MFA2 rD, mT [,n]* Move the contents of the upper halves of accumulator register pair mT to register rD with optional right shift. The rD[31:16] are taken from mTh and rD[15:00] from the corresponding mTl. mTh[31+n: 16+n] || mTl[31+n : 16+n] from the accumulator register pair are transferred to rD[31:00]. The range n = 0 - 8 is permitted for the output alignment shift amount. In the case of n = 0, the field may be omitted.
Divide	*DIVA mD, rS, rT* The contents of register rS is divided by rT, treating the operands as signed 2's complement values. The remainder is sign-extended to 40-bits and stored in mDh and the quotient is sign-extended to 40-bits and stored in mDl. m0h[31:00] is also called HI. m0l[31:00] is also called LO.
Divide Unsigned	*DIVAU mD, rS, rT* The contents of register rS is divided by rT, treating the operands as unsigned values. The remainder is zero-extended to 40-bits and stored in mDh and the quotient is zero-extended to 40-bits and stored in mDl. m0h[31:00] is also called HI. m0l[31:00] is also called LO.
Multiply (32-bit)	*MULTA mD, rS, rT* The contents of register rS is multiplied by rT, treating the operands as signed 2's complement values. The upper 32-bits of the 64-bit product is sign-extended to 40-bits and stored in mDh and the lower 32-bits is zero-extended to 40-bits and stored in the corresponding mDl. m0h[31:00] is also called HI. m0l[31:00] is also called LO. If MMD[MT] is 1, then the partial product rS[15:00] x rT[15:00] is not included in the total product. If MMD[MF] is 1, then the product is left shifted by one bit, and furthermore, if both operands are -1 then the product is set to positive signed, all ones fraction, prior to the shift. If both MMD[MT] and MMD[MF] are 1, the result is undefined.

FIG. 8A

Instruction	Syntax and Description
Multiply Unsigned (32-bit)	*MULTAU mD, rS, rT* The contents of register rS is multiplied by rT, treating the operands as unsigned values. The upper 32-bits of the 64-bit product is zero-extended to 40-bits and stored in mDh and the lower 32-bits is zero-extended to 40-bits and stored in the corresponding mDl. m0h[31:00] is also called HI. m0l[31:00] is also called LO. If MMD[MT] is 1, then the partial product rS[15:00] x rT[15:00] is not included in the total product. If MMD[MF] is 1, then the result is undefined.
Dual Multiply (16-bit)	*MULTA2 {mD, mDh, mDl}, rS, rT* The contents of register rS is multiplied by rT, treating the operands as signed 2's complement values. If the destination register is mDh, rS[31:16] is multiplied by rT[31:16] and the product is sign-extended to 40-bits and stored in mDh. If the destination register is mDl, rS[15:00] is multiplied by rT[15:00] and the product is sign-extended to 40-bits and stored in mDl. If the destination is mD, both operations are performed and the two products are stored in the accumulator register pair mD. If MMD[MF] is 1, then each product is left shifted by one bit, and furthermore, for each multiply, if both operands are -1 then the product is set to positive signed, all ones fraction.
Dual Multiply and Negate (16-bit)	*MULNA2 {mD, mDh, mDl}, rS, rT* The contents of register rS is multiplied by rT, treating the operands as signed 2's complement values. If the destination register is mDh, rS[31:16] is multiplied by rT[31:16] and the product is sign-extended to 40-bits, negated (i.e. subtracted from zero) and stored in mDh. If the destination register is mDl, rS[15:00] is multiplied by rT[15:00] and the product is sign-extended to 40-bits, negated (i.e. subtracted from zero) and stored in mDl. If the destination is mD, both operations are performed and the two products are stored in the accumulator register pair mD. If MMD[MF] is 1, then each product is left shifted by one bit prior to sign-extension and negation, and furthermore, for each multiply, if both operands are -1 then the product is set to positive signed, all ones fraction prior to sign-extension and negation.
Complex Multiply,	*CMULTA mD, rS, rT* rS[31:16] is interpreted as the real part of a complex number. rS[15:00] is interpreted as the imaginary part of the same complex number. Similarly for the contents of general register rT. As the result of CMULTA, mDh is updated with the real part of the product, sign-extended to 40-bits and mDl is updated with the imaginary part of the product, sign-extended to 40-bits. If MMD[MF] is 1, then each product is left shifted by one bit, and furthermore, for each multiply, if both operands are -1 then the product is set to positive signed, all ones fraction, prior to the addition of terms.

800

FIG. 8B

Instruction	Syntax and Description
32-bit Multiply-Add with 72-bit accumulate	*MADDA mD, rS, rT* The contents of register rS is multiplied by rT treating the operands as signed 2's complement values. If MMD[MT] is 1, then the partial product rS[15:00] x rT[15:00] is not included in the total product. If MMD[MF] is 1, then the product is left shifted by one bit, and furthermore, if both operands are -1 then the product is set to a positive signed, all ones fraction. If both MMD[MT] and MMD[MF] are 1, then the result of the multiply is undefined. The 64-bit product is sign-extended to 72-bits and added to the concatenation mDh[39:0] || mDl[31:0], ignoring mDl[39:32]. The lower 32 bits of the result are zero-extended to 40-bits and stored into mDl. The upper 40-bits of the result are stored into mDh.
32-bit unsigned Multiply-Add with 72-bit accumulate	*MADDAU mD, rS, rT* The contents of register rS is multiplied by rT treating the operands as unsigned values. If MMD[MT] is 1, then the partial product rS[15:00] x rT[15:00] is not included in the total product. If MMD[MF] is 1, then the result of the multiply is undefined. The 64-bit product is zero-extended to 72-bits and added to the concatenation mDh[39:0] || mDl[31:0], ignoring mDl[39:32]. The lower 32 bits of the result are zero-extended to 40-bits and stored into mDl. The upper 40-bits of the result are stored into mDh.
Dual Multiply-Add, optional saturation	*MADDA2[.S] {mD, mDh, mDl}, rS, rT* The contents of register rS is multiplied by rT and added to an accumulator register, treating the operands as signed 2's complement values. If the destination register is mDh, rS[31:16] is multiplied by rT[31:16] then sign-extended and added to mDh[39:00]. If the destination register is mDl, rS[15:00] is multiplied by rT[15:00] then sign-extended and added to mDl[39:00]. If the destination is mD, both operations are performed and the two results are stored in the accumulator register pair mD. If MADDA2.S the result of each addition is saturated before storage in the accumulator register. The multiplies are subject to MMD[MF] as in MULTA2. The saturation point is selected as either 40 or 32 bits by MMD[MS].
32-bit Multiply-Subtract with 72-bit accumulate	*MSUBA mD, rS, rT* The contents of register rS is multiplied by rT treating the operands as signed 2's complement values. If MMD[MT] is 1, then the partial product rS[15:00] x rT[15:00] is not included in the total product. If MMD[MF] is 1, then the product is left shifted by one bit, and furthermore, if both operands are -1 then the product is set to a positive signed, all ones fraction. If both MMD[MT] and MMD[MF] are 1, then the result of the multiply is undefined. The 64-bit product is sign-extended to 72-bits and subtracted from the concatenation mDh[39:0] || mDl[31:0], ignoring mDl[39:32]. The lower 32 bits of the result are zero-extended to 40-bits and stored into mDl. The upper 40-bits of the result are stored into mDh.

FIG. 8C

Instruction	Syntax and Description
32-bit unsigned Multiply-Subtract with 72-bit accumulate	*MSUBAU mD, rS, rT* The contents of register rS is multiplied by rT treating the operands as unsigned values. If MMD[MT] is 1, then the partial product rS[15:00] x rT[15:00] is not included in the total product. If MMD[MF] is 1, then the result of the multiply is undefined. The 64-bit product is zero-extended to 72-bits and subtracted from the concatenation mDh[39:0] ‖ mDl[31:0], ignoring mDl[39:32]. The lower 32 bits of the result are zero-extended to 40-bits and stored into mDl. The upper 40-bits of the result are stored into mDh.
Dual Multiply-Sub, optional saturation	*MSUBA2[.S] {mD, mDh, mDl}, rS, rT* The contents of register rS is multiplied by rT and subtracted from an accumulator register, treating the operands as signed 2's complement values. If the destination register is mDh, rS[31:16] is multiplied by rT[31:16] then sign-extended and subtracted from mDh[39:00]. If the destination register is mDl, rS[15:00] is multiplied by rT[15:00] then sign-extended and subtracted from mDl[39:00]. If the destination is mD, both operations are performed and both results are stored in the accumulator register pair mD. If MSUBA2.S the result of each subtraction is saturated before storage in the accumulator register.
Add Accumulators	*ADDMA[.S] mD{h,l}, mS{h,l}, mT{h,l}* The contents of accumulator mTh or mTl is added to the contents of accumulator mSh or mSl, treating both registers as signed 40-bit values. mDh or mDl is updated with the result. If ADDMA.S, the result is saturated before storage. The saturation point is selected as either 40 or 32 bits by MMD[MS].
Subtract Accumulators	*SUBMA[.S] mD{h,l}, mS{h,l}, mT{h,l}* The contents of accumulator mTh or mTl is subtracted from the contents of accumulator mSh or mSl, treating both registers as signed 40-bit values. mDh or mDl is updated with the result. If SUBMA.S, the result is saturated before storage.The saturation point is selected as either 40 or 32 bits by MMD[MS].
Dual Round	*RNDA2 {mT, mTh, mTl} [,n]* The accumulator register mTh or mTl is rounded, then updated. If mT, the accumulator register pair mTh/mTl are each rounded, then updated. The rounding mode is selected in MMD field "RND". The least significant bit of precision in the accumulator register after rounding is: 16+n. Bits [15+n : 00] are zeroed. The range n = 0 - 8 is permitted for the output alignment shift amount. In the case of n = 0, the field may be omitted.

Nomenclature:

rS, rT	=	r0 - r31
mD	=	mDh ‖ mDl; also for mT
mDh	=	m0h - m3h; also for mSh, mTh
mDl	=	m0l - m3l; also for mSh, mTh
HI	=	m0h[31:00]
LO	=	m0l[31:00]

FIG. 8D

Assignment of Instructions of Pipe A, Pipe B

900

	Pipe A	Pipe B
	The Load/Store Pipe	**The MAC Pipe**
MIPS 32-bit General Instructions	MIPS 32-bit General Instructions except: CE1 Custom Engine Opcodes, MULT(U), DIV(U), MFHI, MFLO, MTHI, MTLO,MAD(U),MSUB(U)	MULT(U), DIV(U), MFHI, MFLO, MTHI, MTLO,MAD(U),MSUB(U) CE1 Custom Engine Opcodes, MIPS 32-bit ALU Instructions Note: No Load or Store Instructions
MIPS 32-bit Control Instructions	J, JAL, JR, JALR, JALX SYSCALL, BREAK, All Branch Instructions, All COPz, SWCz, LWCz	
MIPS16 Instructions (No Doubleword Instructions)	All MIPS16 Instructions except: MULT(U), DIV(U), MFHI, MFLO	MULT(U), DIV(U), MFHI, MFLO
EJTAG Instructions	DERET, SDBBP (including MIPS16 SDBBP)	
Lexra Control Instructions	MTRU, MFRU, MTRK, MFRK, MTLXC0,MFLXC0	
Lexra Vector Addressing	LT, ST, LTP, LWP, LHP(U), LBP(U), STP, SWP, SHP, SBP	
Lexra MAC Instructions		MTA2, MFA, MFA2, MULTA, MULTA2, MULNA2, CMULTA, MADDA, MSUBA, ADDMA, SUBMA, DIVA, RNDA2
Lexra Extensions to MIPS ALU Instructions	SLLV2, SRLV2, SRAV2, ADDR, ADDR2, SUBR, SUBR2, SLTR2	SLLV2, SRLV2, SRAV2, ADDR, ADDR2, SUBR, SUBR2, SLTR2
New Lexra ALU Operations	MIN, MIN2, MAX, MAX2, ABSR, ABS2, CLS, MUX2, BITREV, CMVEQZ, CMVNEZ	MIN, MIN2, MAX, MAX2, ABSR, ABS2, CLS, MUX2, BITREV, CMVEQZ, CMVNEZ

FIG. 9

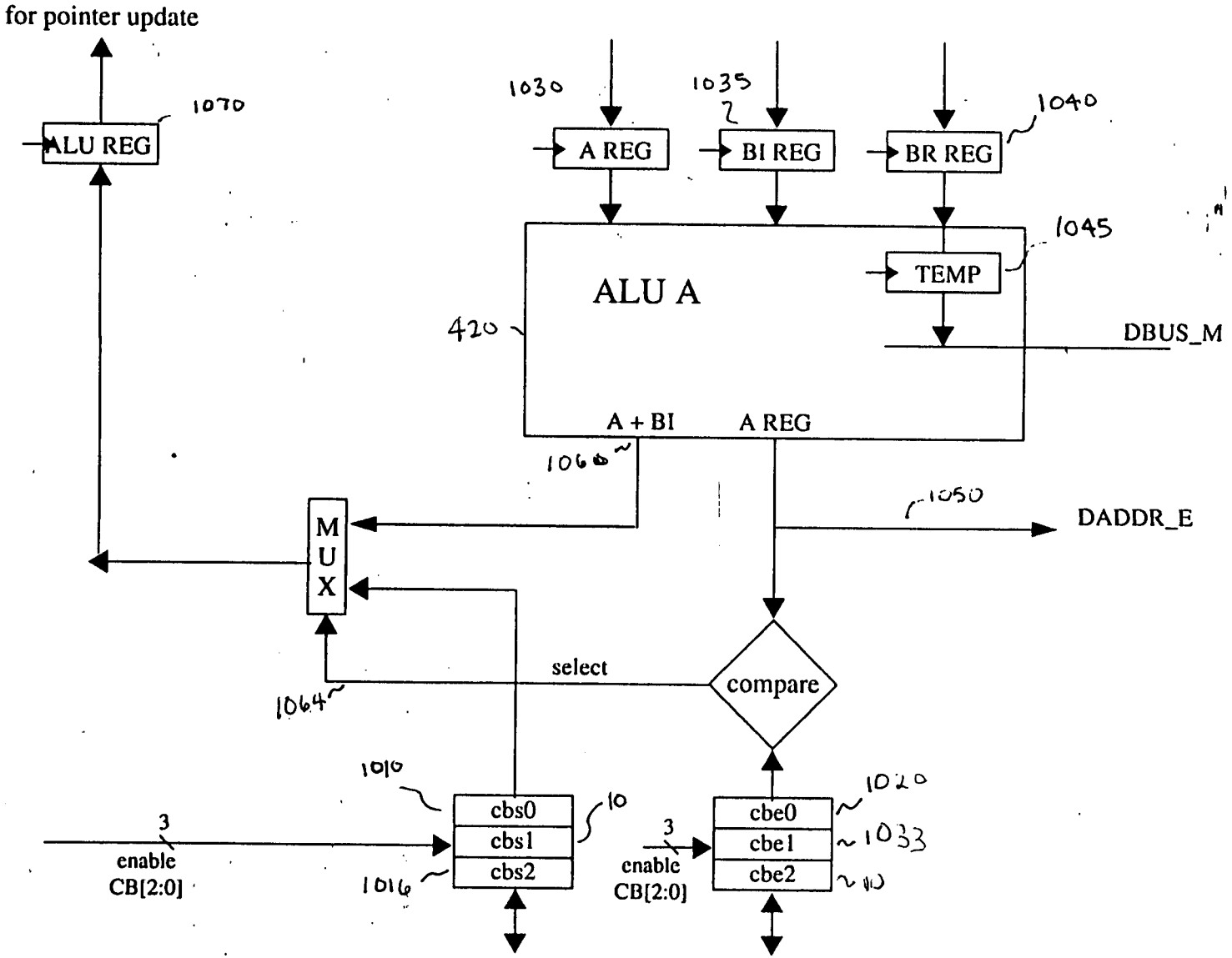


FIG. 10

09637501.081100

Vector Addressing Instruction Summary

1100

Instruction	Syntax and Description
Load Twinword	*LT rT, displacement(base)* The displacement, in bytes, is a signed 14-bit quantity that must be divisible by 8 (since it occupies only 11 bits of the instruction word). Sign-extend the displacement to 32-bits and add to the contents of register *base* to form the address *temp*. Load contents of word addressed by *temp* into register rT (which must be an even register). Load contents of word addressed by *temp*+4 into register rT+1.
Store Twinword	*ST rT, displacement(base)* The displacement, in bytes, is a signed 14-bit quantity that must be divisible by 8 (since it occupies only 11 bits of the instruction word). Sign-extend the displacement to 32-bits and add to the contents of register *base* to form the address *temp*. Store contents of register rT (which must be an even register) into word addressed by *temp*. Store contents of register rT+1 into word addressed by *temp*+4.
Load Twinword, Pointer Increment, optional circular buffer	*LTP[.Cn] rT, (pointer)stride* Let *temp* = contents of register *pointer*. Load contents of word addressed by *temp* into register rT (which must be an even register). Load contents of word addressed by *temp*+4 into register rT+1. The stride, in bytes, is a signed 11-bit quantity that must be divisible by 8 (since it occupies only 8 bits of the instruction word). Sign-extend the stride to 32-bits and add to contents of register *pointer* to form next address. Update *pointer* with the calculated next address. ".Cn" selects circular buffer n = 0 - 2. See Note 2.
Load Word, Pointer Increment, optional circular buffer	*LWP[.Cn] rT, (pointer)stride* Load contents of word addressed by register *pointer* into register rT. The stride, in bytes, is a signed 10-bit quantity that must be divisible by 4 (since it occupies only 8 bits of the instruction word). Sign-extend the stride to 32-bits and add to contents of register *pointer* to form next address.Update *pointer* with the calculated next address. ".Cn" selects circular buffer n = 0 - 2. See Note 2.
Load Halfword, Pointer Increment, optional circular buffer	*LHP[.Cn] rT, (pointer)stride* Load contents of sign-extended halfword addressed by register *pointer* into register rT. The stride, in bytes, is a signed 9-bit quantity that must be divisible by 2 (since it occupies only 8 bits of the instruction word). Sign-extend the stride to 32-bits and add to contents of register *pointer* to form next address. Update *pointer* with the calculated next address. ".Cn" selects circular buffer n = 0 - 2. See Note 2.
Load Halfword Unsigned, Pointer Increment, optional circular buffer	*LHPU[.Cn] rT, (pointer)stride* Load contents of zero-extended halfword addressed by register *pointer* into register rT. The stride, in bytes, is a signed 9-bit quantity that must be divisible by 2 (since it occupies only 8 bits of the instruction word). Sign-extend the stride to 32-bits and add to contents of register *pointer* to form next address. Update *pointer* with the calculated next address. ".Cn" selects circular buffer n = 0 - 2. See Note 2.

FIG. 11A

Vector Addressing Instruction Summary

Instruction	Syntax and Description
Load Byte, Pointer Increment, optional circular buffer	*LBP[.Cn] rT, (pointer)stride* Load contents of sign-extended byte addressed by register *pointer* into register rT. The stride, in bytes, is a signed 8-bit quantity. Sign-extend the stride to 32-bits and add to contents of register *pointer* to form next address. Update *pointer* with the calculated next address. ".Cn" selects circular buffer n = 0 - 2. See Note 2.
Load Byte Unsigned, Pointer Increment, optional circular buffer	*LBPU[.Cn] rT, (pointer)stride* Load contents of zero-extended byte addressed by register *pointer* into register rT. The stride, in bytes, is a signed 8-bit quantity. Sign-extend the stride to 32-bits and add to contents of register *pointer* to form next address. Update *pointer* with the calculated next address. ".Cn" selects circular buffer n = 0 - 2. See Note 2.
Store Twinword, Pointer Increment, optional circular buffer	*STP[.Cn] rT, (pointer)stride* Let *temp* = contents of register *pointer*. Store contents of register rT (which must be an even register) into word addressed by *temp*. Store contents of register rT+1 into word addressed by *temp*+4. The stride, in bytes, is a signed 11-bit quantity that must be divisible by 8 (since it occupies only 8 bits of the instruction word). Sign-extend the stride to 32-bits and add to contents of register *pointer* to form next address.Update *pointer* with the calculated next address. ".Cn" selects circular buffer n = 0 - 2. See Note 2.
Store Word, Pointer Increment, optional circular buffer	*SWP[.Cn] rT, (pointer)stride* Store contents of register rT into word addressed by register *pointer*. The stride, in bytes, is a signed 10-bit quantity that must be divisible by 4 (since it occupies only 8 bits of the instruction word). Sign-extend the stride to 32-bits and add to contents of register *pointer* to form next address. Update *pointer* with the calculated next address. ".Cn" selects circular buffer n = 0 - 2. See Note 2.
Store Halfword, Pointer Increment, optional circular buffer	*SHP[.Cn] rT, (pointer)stride* Store contents of register rT[15:00] into 16-bit halfword addressed by register *pointer*. The stride, in bytes, is a signed 9-bit quantity that must be divisible by 2 (since it occupies only 8 bits of the instruction word). Sign-extend the stride to 32-bits and add to contents of register *pointer* to form next address. Update *pointer* with the calculated next address. ".Cn" selects circular buffer n = 0 - 2. See Note 2.
Store Byte, Pointer Increment, optional circular buffer	*SBP[.Cn] rT, (pointer)stride* Store contents of register rT[07:00] into byte addressed by register *pointer*. The stride, in bytes, is a signed 8-bit quantity. Sign-extend the stride to 32-bits and add to contents of register *pointer* to form next address. Update *pointer* with the calculated next address. ".Cn" selects circular buffer n = 0 - 2. See Note 2.
Move To Radiax, User	*MTRU rT, RADREG* Move the contents of register rT to one of the User Radiax registers: cbs0 - cbs2, cbe0 - cbe2, mmd, lpc0, lpe0, lps0. This instruction has a single delay slot before the updated register takes effect.

FIG 11B-

Instruction	Syntax and Description
Move From Radiax, User	*MFRU rT, RADREG* Move the contents of the designated User Radiax register (cbs0 - cbs2, cbe0 - cbe2, mmd, lpc0, lps0, lpe0) to register rT.

Nomenclature:

rT	=	r0 - r31, and must be even for LT, ST, LTP[.Cn], STP[.Cn]
base, pointer	=	r0 - r31
stride	=	8/9/10/11-bit signed value (in bytes) for byte/halfword/word/twinword ops.
displacement	=	14-bit signed value, in bytes
RADREG	=	cbs0 - cbs2, cbe0 - cbe2, mmd, lpc0, lps0, lpe0

Notes:

1. For LTP[.Cn], LWP[.Cn], LHP(U)[.Cn], LBP(U)[.Cn], rT = *pointer* is unsupported.
2. When a circular buffer is selected, the update of the pointer register is performed according to the following algorithm, which depends on the sign of the stride and the granularity of the access. A stride exactly equal to 0 is not supported:

For LBP(U).Cn and SBP.Cn:

```
     if (stride > 0 && pointer[2:0] == 111 && pointer[31:3] == CBEn)
        then  pointer <= CBSn[31:3] || 000
else if (stride < 0 && pointer[2:0] == 000 && pointer[31:3] == CBSn)
        then  pointer <= CBEn[31:3] || 111
else            pointer <= pointer + stride.
```

For LHP(U).Cn and SHP.Cn

```
     if (stride > 0 && pointer[2:0] == 11x && pointer[31:3] == CBEn)
        then  pointer <= CBSn[31:3] || 000
else if (stride < 0 && pointer[2:0] == 00x && pointer[31:3] == CBSn)
        then  pointer <= CBEn[31:3] || 110
else            pointer <= pointer + stride.
```

For LWP.Cn and SWP.Cn

```
     if (stride > 0 && pointer[2:0] == 1xx && pointer[31:3] == CBEn)
        then  pointer <= CBSn[31:3] || 000
else if (stride < 0 && pointer[2:0] == 0xx && pointer[31:3] == CBSn)
        then  pointer <= CBEn[31:3] || 100
else            pointer <= pointer + stride.
```

For LTP.Cn and STP.Cn

```
     if (stride > 0                          && pointer[31:3] == CBEn)
        then  pointer <= CBSn[31:3] || 000
else if (stride < 0                          && pointer[31:3] == CBSn)
        then  pointer <= CBEn[31:3] || 000
else            pointer <= pointer + stride.
```

FIG. 4C

Extensions to MIPS ALU Operations

1200

Instruction	Syntax and Description
Dual Shift Left Logical Variable	*SLLV2 rD, rT, rS* The contents of rT[31:16] and the contents of rT[15:00] are independently shifted left by the number of bits specified by the low order four bits of the contents of general register rS, inserting zeros into the low order bits of rT[31:16] and rT[15:00]. For SLLV2, the high and low results are concatenated and placed in register rD. (Note that a [.S] option is not provided because this is a *logical* rather than *arithmetic* shift and thus the concept of arithmetic overflow is not relevant.)
Dual Shift Right Logical Variable	*SRLV2 rD, rT, rS* The contents of rT[31:16] and the contents of rT[15:00] are independently shifted right by the number of bits specified by the low order four bits of the contents of general register rS, inserting zeros into the high order bits of rT[31:16] and rT[15:00]. The high and low results are concatenated and placed in register rD. (Note that a [.S] option is not provided because this is a *logical* rather than *arithmetic* shift and thus the concept of arithmetic overflow is not relevant.)
Dual Shift Right Arithmetic Variable	*SRAV2 rD, rT, rS* The contents of rT[31:16] and the contents of rT[15:00] are independently shifted right by the number of bits specified by the low order four bits of the contents of general register rS, sign-extending the high order bits of rT[31:16] and rT[15:00]. The high and low results are concatenated and placed in register rD. (Note that a [.S] option is not provided because arithmetic overflow/underflow is not possible.)
Add, optional saturation	*ADDR[.S] rD, rS, rT* 32-bit addition. Considering both quantities as signed 32-bit integers, add the contents of register rS to rT. For ADDR, the result is placed in register rD, ignoring any overflow or underflow. For ADDR.S, the result is saturated to $0 \| 1^{31}$ (if overflow) or $1 \| 0^{31}$ (if underflow) then placed in rD. ADDR[.S] will not cause an Overflow Trap.
Dual Add, optional saturation	*ADDR2[.S] rD, rS, rT* Dual 16-bit addition. Considering all quantities as signed 16-bit integers, add the contents of register rS[15:00] to rT[15:00] and, independently add the contents of register rS[31:16] to rT[31:16]. For ADDR2, the high and low results are concatenated and placed in register rD ignoring any overflow or underflow. For ADDR2.S, the two results are independently saturated to $0 \| 1^{15}$ (if overflow) or $1 \| 0^{15}$ (if underflow) then placed in rD. ADDR2[.S] will not cause an Overflow Trap.

FIG. 12A

Instruction	Syntax and Description
Subtract, optional saturation	*SUBR[.S] rD, rS, rT* 32-bit subtraction. Considering both quantities as signed 32-bit integers, subtract the contents of register rT from the contents of register rS. For SUBR, the result is placed in register rD ignoring any overflow or underflow. For SUBR.S, the result is saturated to $0 \| 1^{31}$ (if overflow) or $1 \| 0^{31}$ (if underflow) then placed in rD. SUBR[.S] will not cause an Overflow Trap.
Dual Subtract, optional saturation	*SUBR2[.S] rD, rS, rT* Dual 16-bit subtraction. Considering all quantities as signed 16-bit integers, subtract the contents of register rT[15:00] from rS[15:00] and, independently subtract the contents of register rT[31:16] from rS[31:16]. For SUBR2, the high and low results are concatenated and placed in register rD ignoring any overflow or underflow. For SUBR2.S, the two results are independently saturated to $0 \| 1^{15}$ (if overflow) or $1 \| 0^{15}$ (if underflow) then placed in rD. SUBR2[.S] will not cause an Overflow Trap.
Dual Set On Less Than	*SLTR2 rD, rS, rT* Dual 16-bit comparison. Considering both quantities as signed 16-bit integers, if rS[15:00} is less than rT[15:00] then set rD[15:00] to $0^{15} \| 1$, else to zero. Independently, considering both quantities as signed 16-bit integers, if rS[31:16} is less than rT[31:16] then set rD[31:16] to $0^{15} \| 1$, else to zero.

Nomenclature:

rD	=	r0 - r31
rS	=	r0 - r31
rT	=	r0 - r31

1200

FIG. 12B

ALU Operations

1300

Instruction	Syntax and Description
Minimum	*MIN rD, rS, rT* The contents of the general register rT are compared with rS considering both quantities as signed 32-bit integers. If rS < rT or rS = rT, rS is placed into rD. If, rS > rT, rT is placed into rD.
Dual Minimum	*MIN2 rD, rS, rT* The contents of rT[31:16] are compared with rS[31:16] considering both quantities as signed 16-bit integers. If rS[31:16] < rT[31:16] or rS[31:16] = rT[31:16], rS[31:16] is placed into rD[31:16]. If, rS[31:16] > rT[31:16], rT[31:16] is placed into rD[31:16]. A similar, independent operation is performed on rT[15:00] and rS[15:00] to determine rD[15:00].
Maximum	*MAX rD, rS, rT* The contents of the general register rT are compared with rS considering both quantities as signed 32-bit integers. If rS > rT or rS = rT, rS is placed into rD. If, rS < rT, rT is placed into rD.
Dual Maximum	*MAX2 rD, rS, rT* The contents of rT[31:16] are compared with rS[31:16] considering both quantities as signed 16-bit integers. If rS[31:16] > rT[31:16] or rS[31:16] = rT[31:16], rS[31:16] is placed into rD[31:16]. If, rS[31:16] < rT[31:16], rT[31:16] is placed into rD[31:16]. A similar, independent operation is performed on rT[15:00] and rS[15:00] to determine rD[15:00].
Absolute, optional saturation	*ABSR[.S] rD, rT* Considering rT as a signed 32-bit integer, if rT > 0, rT is placed into rD. If rT < 0, -rT is placed into rD. If ABSR.S and rT = $1 \| 0^{31}$ (the smallest negative number) then $0 \| 1^{31}$ (the largest positive number) is placed into rD; otherwise, if ABSR and rT = $1 \| 0^{31}$, rT is placed into rD.
Dual Absolute, optional saturation	*ABSR2[.S] rD, rT* ABS[.S] operations are performed independently on rT[31:16] and rT[15:00], considering each to be 16-bit signed integers. rD is updated with the absolute value of rT[31:16] concatenated with the absolute value of rT[15:00].
Dual Mux	*MUX2[[.HH], [.HL], [.LH], [.LL]] rD, rS, rT* rD[31:16] is updated with rS[31:16] for MUX2.HH or MUX2.HL. rD[31:16] is updated with rS[15:00] for MUX2.LH or MUX2.LL. rD[15:00] is updated with rT[31:16] for MUX2.HH or MUX2.LH rD[15:00] is updated with rT[15:00] for MUX2.HL or MUX2.LL
Count Leading Sign bits	*CLS rD, rT* The binary-encoded number of redundant sign bits of general register rT is placed into rD. If rT[31:30] = 10 or 01, rD is updated with 0. If rT = 0, or if rT = 1^{32}, rD is updated with $0^{27} \| 1^{5}$ (decimal 31).

FIG. 13A

ALU Operations

Instruction	Syntax and Description
Bit Reverse	*BITREV rD, rT, rS* A bit-reversal of the contents of general register rT is performed. The result is then shifted right logically by the amount specified in the lower 5-bits of the contents of general register rS, then stored in rD.

1300

Nomenclature:

rD	=	r0 - r31
rS	=	r0 - r31
rT	=	r0 - r31

0508050-0508050

FIG. 13B

Conditional Operations

1400

Instruction	Syntax and Description
Conditional Move on Equal Zero	*CMVEQZ[.H] [.L] rD, rS, rT* If the general register rT is equal to 0, the general register rD is updated with rS; otherwise rD is unchanged. For [.H] if rT[31:16] is equal to 0, the *full 32-bit* general register rD[31:00] is updated with rS; otherwise rD is unchanged. For [.L] if rT[15:00] is equal to 0, the *full 32-bit* general register rD[31:00] is updated with rS; otherwise rD is unchanged.
Conditional Move on Not Equal Zero	*CMVNEZ[.H] [.L] rD, rS, rT* If the general register rT is not equal to 0, the general register rD is updated with rS; otherwise rD is unchanged. For [.H] if rT[31:16] is not equal to 0, the *full 32-bit* general register rD[31:00] is updated with rS; otherwise rD is unchanged. For [.L] if rT[15:00] is not equal to 0, the *full 32-bit* general register rD[31:00] is updated with rS; otherwise rD is unchanged.

Nomenclature:

rD	=	r0 - r31
rS	=	r0 - r31
rT	=	r0 - r31

Usage Note:
When combined with the SLT or SLTR2 instructions, the conditional move instructions can be used to construct a complete set of conditional move macro-operations. For example:

```
if ( r3 < r4 )   r1 <-- r2

  CMVLT   r1,r2,r3,r4      ===> SLT      AT,r3,r4
                                CMVNEZ    r1,r2,AT
if ( r3 >= r4 )  r1 <-- r2

  CMVGE   r1,r2,r3,r4      ===> SLT      AT,r3,r4
                                CMVEQZ    r1,r2,AT
if ( r3 <= r4 )  r1 <-- r2

  CMVLE   r1,r2,r3,r4      ===> SLT      AT,r4,r3
                                CMVEQZ    r1,r2,AT
f ( r3 > r4 )    r1 <-- r2

  CMVGT   r1,r2,r3,r4      ===> SLT      AT,r4,r3
                                CMVNEZ    r1,r2,AT
```

FIG. 14

Cycles Required Between Dual MAC Instructions

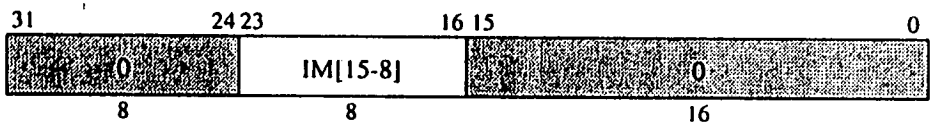
1500

1st Op / 2nd Op	MULTA(U)	MADDA(U), MSUBA(U)	CMULTA	DIVA(U)	MADDA2[.S], MSUBA2[.S], ADDMA[.S], SUBMA[.S], MULTA2, MULNA2, RNDA2, MTA2	MFA
MULTA(U), MADDA(U),MSUBA(U)	1U	1U	1U	(19T)	-	-
DIVA(U)	(3T)	(4T)	(1T)	19U	-	-
CMULTA, MADDA2[.S], MSUBA2[.S], MULTA2, MULNA2, MTA2	3U	4U	1U	(19T)	-	-
ADDMA[.S], SUBMA[.S], RNDA2	LO 2S 2T / HI 3S 3T	4U	1S 1T	(19S) (19T)	-	-
MFA	LO 4S / HI 5S	LO 5S / HI 6S	3S	19S	2S	-

Notes:
- means the two ops can be issued back-to-back.
xU indicates unconditional delay of the indicated number of cycles.
xS indicates delay only if (any) 2nd Op source is the same as (any) 1st Op target (producer-consumer dependency).
xT indicates delay only if (any) 2nd Op target is the same as (any) 1s Op target (preserve write after write order).
Items in parenthesis are unlikely to occur in any useful program, which would probably have an intervening MFA.
LO/HI indicate that for the 72-bit result of a 32x32 MULT or MADDA, the LO 32-bits (m0l, m1l, etc.) are available one cycle earlier.
Delay of "x" cycles means that if the 1st Op issues in cycle N, then the 2nd Op may issue in cycle N+x+1.

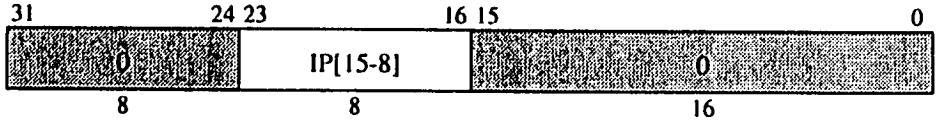
FIG.15

New: ESTATUS (LX COP0 reg 0) Read/Write



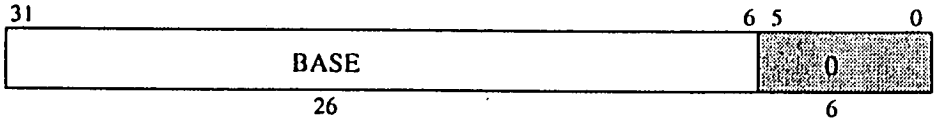
1610

New: ECAUSE (LX COP0 reg 1) Read-only



1620

New: INTVEC (LX COP0 reg 2) Read/Write



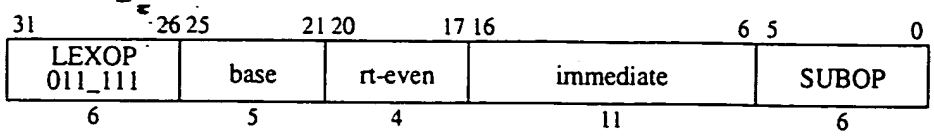
1630

IM[15-8] is reset to 0.

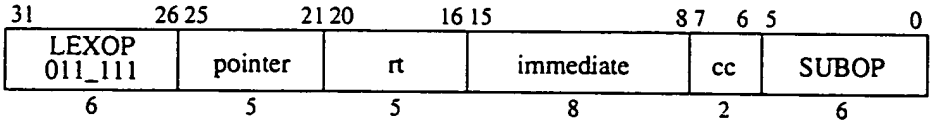
FIG. 16

09637500 .081100

I. Load/Store Formats



Assembler Mnemonic	base	rt-even	immediate	Lexra SUBOP
LT	base	rt-even	displacement/8	LT
ST	base	rt-even	displacement/8	ST

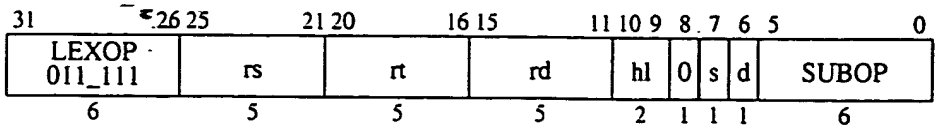


Assembler Mnemonic	pointer	rt	immediate	cc	Lexra SUBOP
LBP[.Cn]	pointer	rt	stride	cc	LBP
LBPU[.Cn]	pointer	rt	stride	cc	LBPU
LHP[.Cn]	pointer	rt	stride/2	cc	LHP
LHPU[.Cn]	pointer	rt	stride/2	cc	LHPU
LWP[.Cn]	pointer	rt	stride/4	cc	LWP
LTP[.Cn]	pointer	rt	stride/8	cc	LTP
SBP[.Cn]	pointer	rt	stride	cc	SBP
SHP[.Cn]	pointer	rt	stride/2	cc	SHP
SWP[.Cn]	pointer	rt	stride/4	cc	SWP
STP[.Cn]	pointer	rt	stride/8	cc	STP

base, pointer, rt — Selects general register r0 - r31.
rt-even — Selects general register even-odd pair r0/r1, r2/r3, ... r30/r31
stride — Signed 2s-complement number in bytes. Must be an integral number of halfwords/words/twinwords for the corresponding instructions.
displacement — Signed 2s-complement number in bytes. Must be an integral number of twinwords.
cc
00 — select circular buffer 0 (cbs0, cbe0)
01 — select circular buffer 1 (cbs1, cbe1)
10 — select circular buffer 2 (cbs2, cbe2)
11 — no circular buffer selected

FIG. 17A

II. Arithmetic Format



Assembler Mnemonic	rs	rt	rd	hl	s	d	Lexra SUBOP
ADDR[.S],ADDR2[.S]	rs	rt	rd	0	s	d	ADDR
SUBR.S, SUBR2[.S]	rs	rt	rd	0	s	d	SUBR
SLTR2	rs	rt	rd	0	0	1	SLTR
SLLV2	rs	rt	rd	0	0	1	SLLV
SRLV2	rs	rt	rd	0	0	1	SRLV
SRAV2	rs	rt	rd	0	0	1	SRAV
MIN, MIN2	rs	rt	rd	0	0	d	MIN
MAX, MAX2	rs	rt	rd	0	0	d	MAX
ABSR[.S], ABSR2[.S]	0	rt	rd	0	s	d	ABSR
MUX2.[LL,LH,HL,HH]	rs	rt	rd	hl	0	1	MUX
CLS	0	rt	rd	0	0	0	CLS
BITREV	rs	rt	rd	0	0	0	BITREV

rs, rt, rd

Selects general register r0 - r31.

s

Selects saturation of result. s=1 indicates that saturation is performed.

d

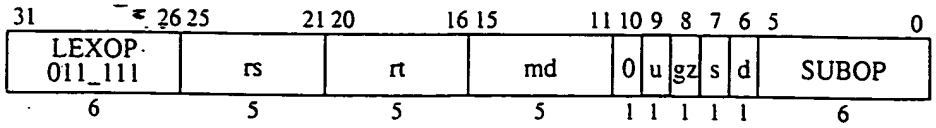
d=1 indicates that dual operations on 16-bit data are performed.

hl (for MUX2)

00 LL: rD = rs[15:00] || rt[15:00]
01 LH: rD = rs[15:00] || rt[31:16]
10 HL: rD = rs[31:16] || rt[15:00]
11 HH: rD = rs[31:16] || rt[31:16]

FIG. 17B

III. MAC Format A



Assembler Mnemonic	rs	rt	md	u	gz	s	d	Lexra SUBOP
CMULTA	rs	rt	md	0	0	0	0	CMULTA
DIVA(U)	rs	rt	md	u	0	0	0	DIVA
MULTA(U)	rs	rt	md	u	1	0	0	MADDA
MULTA2	rs	rt	md	0	1	0	1	MADDA
MADDA(U)	rs	rt	md	u	0	0	0	MADDA
MADDA2[.S]	rs	rt	md	0	0	s	1	MADDA
MSUBA(U)	rs	rt	md	u	0	0	0	MSUBA
MSUBA2[.S]	rs	rt	md	0	0	s	1	MSUBA
MULNA2	rs	rt	md	0	1	0	1	MSUBA
MTA2[.G]	rs	0	md	0	g	0	1	MTA

rs, rt — Selects general register r0 - r31.

md — Selects accumulator, 0NNHL where,

NN = m0 - m3
HL
00 = reserved — 01 = mNl
10 = mNh — 11 = mN

s — Selects saturation of result. s=1 indicates that saturation is performed.

d — d=1 indicates that dual operations on 16-bit data are performed.

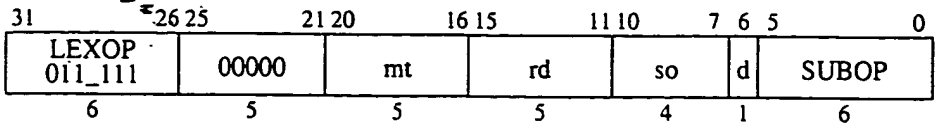
gz — For MTA2, used as "guard" bit. If g=1, bits [39:32] of the accumulator (pair) are loaded and bits [31:00] are unchanged. If g=0, all 40 bits [39:00] of the accumulator (or pair) are updated.

For MADDA, MSUBA, used as a "zero" bit. If z = 1, the result is added to (subtracted from) zero rather than the previous accumulator value; this performs a MULTA, MULTA2 or MULNA2. If z = 0, performs a MADDA, MSUBA, MADDA2 or MSUBA2.

u — Treat operands as unsigned values (0 = signed, 1 = unsigned)

FIG. 17C

IV. MAC Format B



Assembler Mnemonic	mt	rd	so	d	Lexra SUBOP
MFA, MFA2	mt	rd	so	d	MFA
RNDA2	mt	0	so	1	RNDA

rd

Selects general register r0 - r31.

mt Selects accumulator, 0NNHL where,

NN = m0 - m3
HL
00 = reserved
01 = mNl
10 = mNh
11 = mN

d

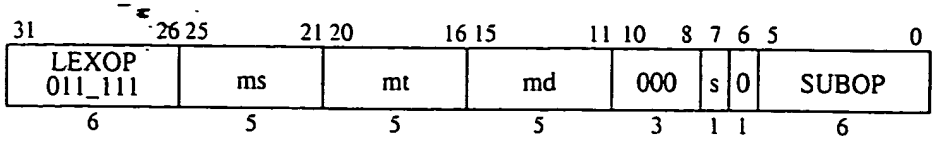
d=1 indicates that dual operations on 16-bit data are performed.

so

Encoded ("output") shift amount n = 0 - 8 for RNDA2, MFA, MFA2 instructions.

FIG. 17D

V. MAC Format C



Assembler Mnemonic	ms	mt	md	s	Lexra SUBOP
ADDMA[.S]	ms	mt	md	s	ADDMA
SUBMA[.S]	ms	mt	md	s	SUBMA

mt, ms, md

Selects accumulator, 0NNHL where,

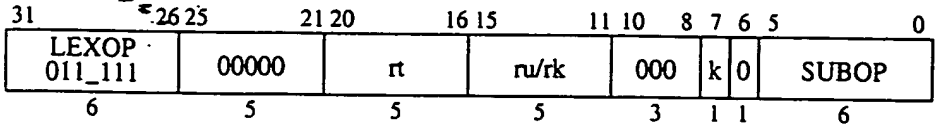
NN = m0 - m3
HL
00 = reserved
01 = mNl
10 = mNh
11 = reserved

s

Selects saturation of result. s=1 indicates that saturation is performed.

FIG. 17E

VI. RADIAX MOVE Format and Lexra-Cop0 MTLXC0/MFLXC0 Instructions



Assembler Mnemonic	rt	ru/rk	k	Lexra SUBOP
MFRU	rt	ru	0	MFRAD
MTRU	rt	ru	0	MTRAD
MFRK	rt	rk	1	MFRAD
MTRK	rt	rk	1	MTRAD

rt

Selects general register r0 - r31.

rk

Selects Radiax Kernel register in MFRK, MTRK instructions — currently all reserved. However, a Coprocessor Unusable Exception is taken in User mode if the Cu0 bit is 0 in the CP0 Status register when MFRK or MTRK is executed.

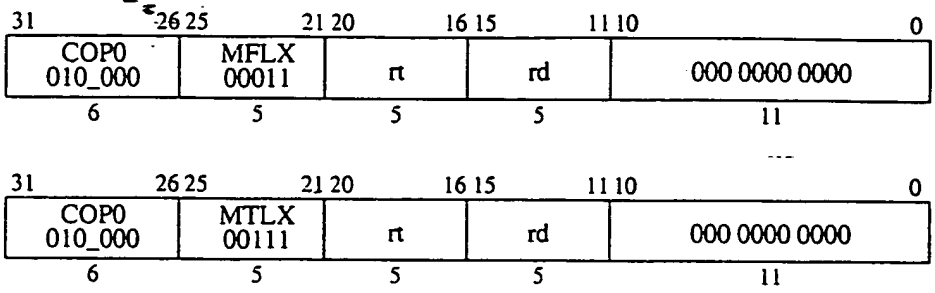
ru

Selects Radiax User register in MFRU, MTRU instructions.

00000 cbs0
00001 cbs1
00010 cbs2
00011 reserved
00100 cbe0
00101 cbe1
00110 cbe2
00111 reserved
01xxx reserved
10000 lps0
10001 lpe0
10010 lpc0
10011 reserved
101xx reserved
11000 mmd
11001 reserved
111xx reserved

FIG. 17F

Lexra-Coprocessor0 Register Access Instructions



Assembler Mnemonic	Copz rs	rt	rd
MFLXC0	MFLX	rt	rd
MTLXC0	MTLX	rt	rd

These are *not* LEXOP instructions. They are variants of the standard MTC0 and MFC0 instructions that allow access to the Lexra Coprocessor0 Registers listed below. As with any COP0 instruction, a Coprocessor Unusable Exception is taken in User mode if the Cu0 bit is 0 in the CP0 Status register when these instructions are executed.

rt

Selects general register r0 - r31.

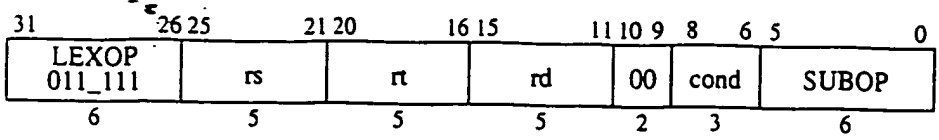
rd

Selects Lexra Coprocessor0 register:

00000 ESTATUS
00001 ECAUSE
00010 INTVEC
00011 reserved
001xx reserved
01xxx reserved
1xxxx reserved

FIG. 17G

VII. CMOVE Format



Assembler Mnemonic	rs	rt	rd	cond	Lexra SUBOP
CMVEQZ[.H][.L]	rs	rt	rd	cond	CMOVE
CMVNEZ[.H][.L]	rs	rt	rd	cond	CMOVE

rs, rt, rd

Selects general register r0 - r31.

cond

Condition code for rT operand referenced by the conditional move.

000 EQZ
001 NEZ
010 EQZ.H
011 NEZ.H
100 EQZ.L
101 NEZ.L
11x reserved

FIG. 17H